1 VSCode

Prinzipiell ist es egal, mit welcher IDE man arbeitet. Da VSCode jedoch sehr flexibel und im Webentwicklungsbereich durchaus verbreitet ist, bezieht sich dieser Kurs immer auf den Umgang mit VSCode. Das Programm VSCode kann unter https://code.visualstudio.com/ geladen und installiert werden. Hier findet man unter „Download“ die Versionen des Programms. Wer möchte, dass auf seinem System alle angemeldeten User VSCode nutzen können, sollte unter „Other platforms“ den „System Installer“ wählen. Danach werden die Standardvorschläge für die Einstellungen übernommen und das Programm installiert. Auf den Schulrechnern ist VSCode bereits vorhanden.

2 Node.js

JavaScript wurde ursprünglich für die Dynamisierung von html-Seiten erstellt. Mit der Zeit wurde aus dieser relativ einfach gestrickten Programmiersprache ein vollumfängliches Werkzeug, welches dank JIT Compiler auch sehr performant agiert. Die im Chrome Browser verbaute Scripting-Engine namens „V8“ wurde von Google als open Source Projekt geführt, weshalb man in der Lage war, diese Engine als Modul in anderen Softwareprojekten einzusetzen. Das bekannteste hierfür ist „node.js“, was eine serverseitige Nutzung von JavaScript erlaubt. Hieraus ergeben sich kleinere Unterschiede zur browserseitigen Nutzung:

- Zugriff auf das Filesystem ist möglich
- Mangels html-Seite gibt es keinen Zugriff auf das DOM
- Die Komponenten für Multithreading und paralleler Abarbeitung von I/O sind auf eventuelle Einsätze als Server angepasst worden
- Node.js verfügt über einen „Node Package Manager“, kurz npm. Dieser kümmert sich um das Bereitstellen / Installieren von Paketen
- Weiterhin können mit node.js über „`require`“ Module geladen werden

Die Laufzeitumgebung node.js kann unter https://nodejs.org/en/download/ heruntergeladen und anschließend installiert werden. Hierbei übernehmen wir die einzelnen Einstellungen und wählen zusätzlich noch die Installation der „notwendigen Tools“ aus. Dadurch wird im Anschluss an die node.js Installation noch ein weiteres Fenster geöffnet, in dem die Zusatztools auf den Rechner gebracht werden. Danach empfiehlt der Installer einen Reboot. Auf den Schulrechnern ist node.js bereits vorhanden.

Bevor wir mit node.js beginnen, noch ein kurzer Hinweis auf den npm. Dieser verwaltet Codepakete, so dass sie im Zusammenhang mit anderen Paketen möglichst einfach zusammenspielen. Details zum Package Manager finden wir unter https://nodejs.dev/learn/an-introduction-to-the-npm-package-manager.

Das wichtigste File des npm ist das package.json File. Dort werden alle für das Projekt notwendigen Informationen gehalten – inklusive Abhängigkeiten zu anderen Bibliotheken. Wenn wir ein Projekt starten, so benötigen wir eben dieses File. Am einfachsten wird es erstellt, indem wir ein eigenes (Projekt-)Verzeichnis erstellen und dieses öffnen. In VSCode geht das bspw. mit einem neuen Terminal.

Anmerkung: der Einfachheit wird hier das „C:\temp“ Verzeichnis genutzt. Für Ihr Projekt sollten Sie natürlich ein anderes Verzeichnis wählen.

```
PS C:\temp> mkdir myServer
PS C:\temp> cd myServer
```

Nun initialisieren wir unser Projekt mit

```
PS C:\temp\myServer> npm init
```

Die Initialisierungsroutine erwartet nun diverse Vorgaben, bei denen wir im Regelfall die Defaultwerte übernehmen können.

Information:	**Wert:**
package name	myserver
version	1.0.0
description	Erster Versuch eines Express Servers in node.js
entry point	app.js
test command	node app.js
git repository	
keywords	
author	Max Mustermann
license	ISC

Danach wird das `package.json` zur Kontrolle angezeigt. Nach der Bestätigung macht npm zwei Dinge:

1. Schreiben des `package.json` Files
2. Jede zusätzliche Installation (bspw. des Express Servers) wird nun in ein Verzeichnis „`node_modules`“ eingetragen und in `package.json` unter `"dependencies": {}` festgehalten.

3 Express

Express ist ein Framework für die Entwicklung von Web-Applikationen, geschrieben in JavaScript. Es ist offen und unterliegt der Creative Commons Lizenz (BY-SA). Der Express Server ist unter anderem in der Lage, folgende Services anzubieten

- Hosting von statischen Webinhalten
- serverseitiges Rendering
- Kommunikation als http Server
- Routen von http Endpunkten
- Kommunikation als Websocket Server

Damit wir den Server nutzen können, müssen wir ihn erstmal über npm installieren:

```
PS C:\temp\myServer> npm install express
```

Nun sind drei Dinge durchgeführt worden:

1. Alle für Express notwendigen Files sind nach *C:\temp\myServer\node_modules* kopiert worden
2. In package.json wurde der Eintrag für Express hinzugefügt
3. Es wurde das File package-lock.json erzeugt, in dem jeweils die Version aller installierten (Sub-)Pakete hinterlegt wurden

Die Version von Express wurde nun an zwei Stellen hinterlegt – package.json und package-lock.json. Die Versionsnummern sind identisch. Prüfen Sie dies und tragen die Version hier ein:

Nun testen wir den Server mit der einfachsten denkbaren Anwendung im Netz – als Server für statische Inhalte. Hierzu öffnen wir mit VSCode den Ordner `C:\temp\myServer` und erstellen das File `app.js`. Die statischen Inhalte legen wir in einem Unterverzeichnis namens „`public`“ ab:

```
PS C:\temp\myServer> mkdir public
```

In diesem Verzeichnis platzieren wir eine einfache HTML-Datei namens „`hello.html`“ mit beliebigem Inhalt, bspw.:

```
<!DOCTYPE html>
<html>
<head>
  <title>My first express content</title>
</head>
<body>
  Hello, World!
</body>
</html>
```

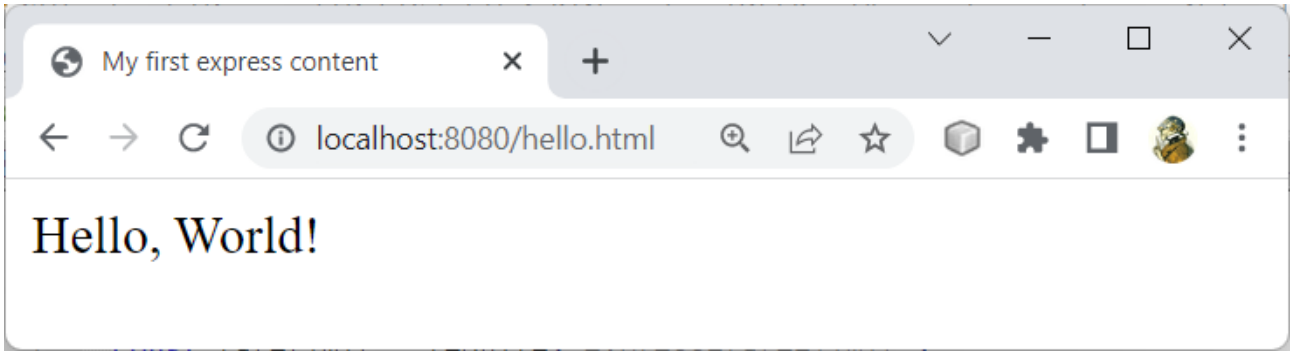
Nun müssen wir den Express Server dazu bringen, die Inhalte tatsächlich an den anfragenden weiterzugeben. Hierzu erzeugen wir unter `C:\temp\myServer` das File `app.js` und tragen folgenden Code ein:

```
// Laden der Funktion zur Erstellung eines Express Servers und Zuweisung
// an eine Konstante "express"
const express = require('express');

// Erstellung des Express Servers und Zuweisung an eine Konstante "app"
const app = express();

// Festlegung des Ports in einer Konstanten für späteren Gebrauch
const EXP_PORT = 8080;

// Erlauben des statischen Zugriffs auf den Unterordner "public", indem
// als Middleware die Methode "static" eingesetzt wird
app.use(express.static('public'));

// Starten des Servers und Übergabe einer Funktion über einen Lambda
// Ausdruck, welche nach Start ausgeführt werden soll
app.listen(EXP_PORT, () => {
    // Konsolenausgabe
    console.log("Ich höre auf Port " + EXP_PORT);
})
```

Nun können wir unseren Server testen. Eine Möglichkeit ist es, den Server direkt als node-Applikation zu starten:

```
PS C:\temp\myServer> node app.js
```

Auf der Konsole sehen wir: `Ich höre auf Port 8080`. Die Adresse http://localhost:8080/hello.html zeigt uns nun auch die einfache HTML Seite an:



Mit CTRL+C (bzw. STRG+C) können wir den Server wieder stoppen. Eine zweite Möglichkeit ist es, den Start über npm durchzuführen, da wir in package.json folgenden Eintrag haben:

```
"scripts": {
    "test": "node app.js"
  },
```

Unter „scripts“ können wir nun beliebige Befehle hinterlegen, welche uns bei der Entwicklung helfen. Weiterhin ist es möglich, mit zusätzlichen Schlüsselwörtern die Skriptausführung auf verschiedene Szenarien anzupassen (bspw. Pre- und Postscripting). Unser erstes Skript wird mit `test` aufgerufen und führt den Befehl „`node app.js`“ aus. Starten wir unser Programm also mit npm:

```
PS C:\temp\myServer> npm test
```

Auch hier startet wieder unser Server, wobei noch ein paar Zusatzinfos ausgegeben werden.

Ein sehr praktisches Tool ist nodemon. Dieses muss zuerst installiert werden (`-g` ist für die globale Installation – also nicht nur im Rahmen des Projektes):

```
PS C:\temp\myServer> npm install -g nodemon
```

Nodemon führt nun genauso wie der node Befehl unser `app.js` aus, prüft aber während der Laufzeit, ob sich relevante Files geändert haben. Wenn ja, so führt nodemon einen Restart aus. Starten wir also unser Programm nochmal mit nodemon:

```
PS C:\temp\myServer> nodemon app.js
```

Der Server startet wieder mit der Ausgabe `Ich höre auf Port 8080`. Nun ändern wir im Code die letzte Zeile wie folgt ab:

```
    console.log("I am listening to port: " + EXP_PORT);
```

Sobald wir gespeichert haben, sehen wir automatisch auf der Konsole den Text `I am listening to port: 8080`.

4 Aufgabenstellung

Ergänzen Sie ihre HTML Seite um ein `css/hello.css` und ein `js/hello.js` File, binden es in die HTML Seite ein und realisieren eine einfache Funktionalität – bspw. ein Button mit „hello World“ Popup und eine Hintergrundfarbe des Bodys über CSS.

Geben Sie an, welche statischen Inhalte in Webapplikationen üblich sind: